COVER STORY

PART 2

A Discipline Seeks to Grasp the Fleeting Silicon Logician

SUMMARY: From its embryonic days in the 1950s, artificial intelligence has evolved into a sophisticated field of scientific inquiry. Early attempts at making a thinking machine, which focused on rules of reason, have given way to more inductive approaches. The expert system was one result. Other efforts pursued systems that could learn, leading to an examination of how language itself is understood.

As its own field of inquiry, artificial intelligence was born on the lawns of Dartmouth College during the summer of 1956.

There, an elite band of scientists met to discuss "learning or any other features of intelligence," as the proposal for the six-week conference stated, so that "a machine can be made to simulate it." With funds from the Rockefeller Foundation — a mere $7,500 — psychologists, mathematicians and engineers began to unfold the logical underpinnings of thought.

The prime movers of this meeting were John McCarthy, a young mathematician at Dartmouth; Marvin Minsky, a Harvard junior fellow in mathematics and neurology; Nathaniel Rochester, a computer scientist at International Business Machines Corp.; and Claude Shannon, a mathematician at Bell Telephone Laboratories Inc., all of whom would go on to make substantial contributions to the field of computers and cognition.

Much talked about at Dartmouth was the work of Herbert Simon and Allen Newell, who only six months earlier had developed a "thinking machine." It was an impressive device, although it did not really think. Logic Theorist, as it was called, was a computer program that used rules of reason to prove theorems in symbolic logic. Its novelty was its ability to deviate from the hard-cut paths typical of most programs. Once it found a better proof for a theorem than one devised by Alfred North Whitehead and Bertrand Russell, two leading early 20th century logicians.

But Logic Theorist's main contribution may have been what Simon and Newell learned while inventing it. A trial-and-error style of problem solving, crucial to scientific discovery, has led many a researcher in artificial intelligence up poor paths, some dead ends. The discipline's history since 1956 is filled with fits and starts, ideas begetting ideas, a pattern true of individuals as well as the entire effort to get machines to "really think."

Simon (left) and Newell, pioneers in logic machines, emphasized problem solving.

DAN McCOY / BLACK STAR

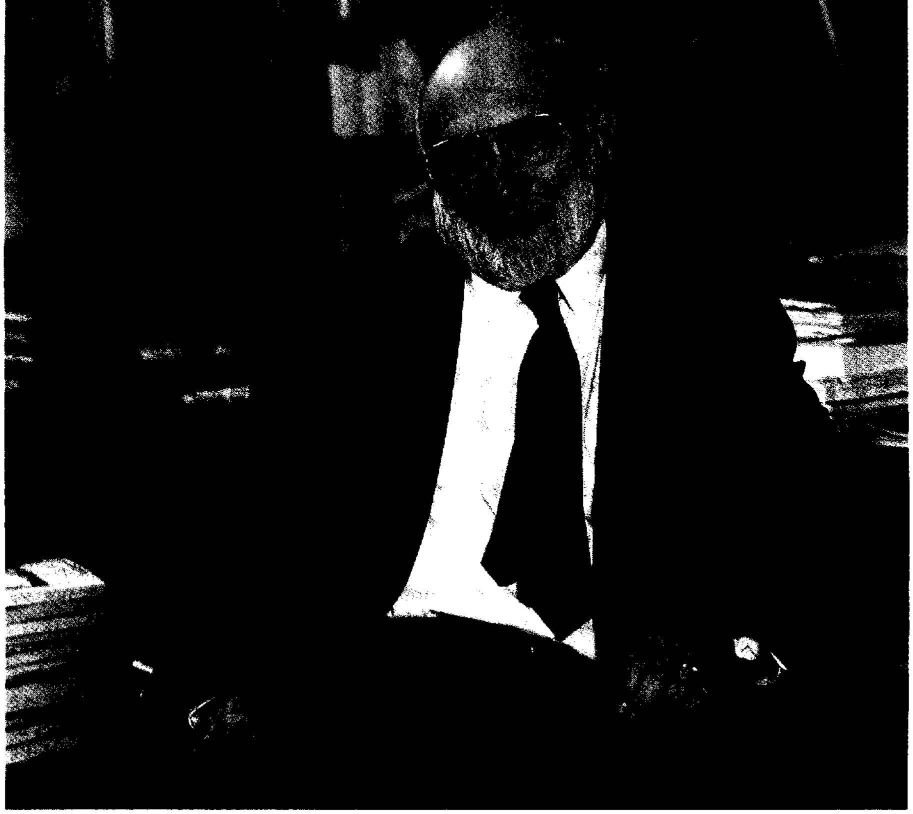
If nothing else, Logic Theorist proved a fundamentally new point: Logic is not so logical. Facing any tough problem — playing chess, planning a trip, solving an equation — creates for the solver a great number of possible options. In chess, there are more possible moves than there are stars in the universe. A master, clearly, has tricks to keep him from weighing out every one. So Simon and Newell devised search-limiting rules to zero in on good solutions, called heuristics, derived from the Greek word for discovery.

By 1957, Logic Theorist gave rise to an improved version, General Problem Solver. It used common human tricks, such as backward reasoning — a method used by, say, sailors, envisioning their destination before setting out to navigate choppy waters. Another trick was "hill climbing," a sort of self-monitoring mechanism to tell a computer when it is getting warm, or nearing a good solution. The metaphor, invented by outdoorsmen, is a method used by those hiking uphill in a fog. Unable to see far ahead, they search out paths that appear to lead up instead of ones that appear to head down. Similarly, the computer would scout fruitful paths of reason.

Unfortunately, one big problem kept blocking progress: Only the surface of reasoning is rational.

Simple statements, such as "Mary gives John a book," involve a nearly infinite set of logical assumptions. John and Mary must be within arm's reach. They are probably in the same room. The book is light enough so that Mary and John can hold it. But what if Mary and John were not in the same room and she still gave the book to John? Well, then, one has to turn to alternative, or default, assumptions. Maybe she mailed it to him. The extent of these assumptions and the number of possible exceptions, known as counterfactuals, are mind-boggling. No intelligent machine, or person, could function if it had to examine every detail of every situation.

Many researchers changed direction, moving from the so-called bottom-up approach — trying to make virtual electronic brain cells — to a top-down approach: devising clever problem-solving systems. That trend owes much to a lucky accident



JOYCE RAVID

Lederberg collaborated on first expert system, a decoder of complex molecules.

in computer programming. In 1959 Herbert Gelernter, a young physicist at IBM, had designed a program to analyze geometric forms. Unaided, it found a novel proof of a mathematical theorem, a cleverer method than the one used by Euclid, the Greek geometer, some 2,300 years ago.

But pure reasoning programs were still, for the most part, getting hung up on trivial details. Often they would get lost inside their own loopy programs or come up with silly generalizations. Well aware of this was Edward Feigenbaum, a Stanford University computer scientist, who began to conclude that, in artificial intelligence, it was "better to be knowledgeable than smart."

In his eyes, sweeping theories about cognition were not panning out very well. They left too much about thinking unexplained. Taking a somewhat different tack, he focused on empirical induction, that is, how people look at groups of objects and infer things about them. The result of his efforts to get computers to think empirically was the expert system, those one-subject-at-a-time reasoners that are so common in business today.

Touted as the father of the expert system, Feigenbaum has devoted much of his career to the would-be silicon Socrates. His book "The Rise of the Expert Company," about expert systems in businesses — "the adolescence of my technological child," he adds — is forthcoming in August.

A chance occurrence carried his research that way. Trained in cognitive psychology but made enthusiastic about artificial intelligence by Simon, with whom he studied, Feigenbaum headed to California 30 years ago to meet Joshua Lederberg, a young organic chemist. Lederberg, now a Nobel laureate who heads Rockefeller University, wanted to use artificial intelligence techniques to decode big molecules.

The result of their collaboration was Dendral, the world's first expert system. It analyzed complex organic molecules. Years later, after much refinement, not only would Dendral unravel molecules as well as a good chemist could, but it would be cited as a contributor in some important technical papers.

Encouraged by Dendral's success, Edward Shortliffe of Stanford Medical School created Mycin, an expert system to advise doctors on the selection of antibiotics for their patients. It could draw conclusions about infections from facts about the patient. More important than that system itself, though, was a discovery made about expert systems in general.

Shortliffe found that Mycin's knowledge base could be separated from its logic mechanism, or inference engine. Entire bases of knowledge, big sets of facts, could be plugged in, so to speak, to the system's logic machine. Thereupon followed Emycin, for Empty Mycin, a factless logic machine.

Now called an expert system shell, an all-purpose reasoning device into which useful facts can be programmed, such systems have great appeal in the marketplace. An early one that worked out well was Caduceus, an artificial doctor. With facts for about 700 diseases on file, the program could diagnose illnesses almost as well as a trained physician.

Yet expert systems, useful as they are, still have a handicap: They have difficulty learning. Any intelligent entity, person or machine, has to be able to learn by example as well as by rote, to learn from its own mistakes. But how?

During the late 1950s, an electrical engineer at IBM, Arthur Samuel, decided to teach a computer to play checkers. Working by day on the company's high-powered 701 computer, by night on his checkers program, Samuel taught his pet program to learn from its own experiences. First by rote, later by generalization, it learned quickly and soared to the level of master. In 1962 Samuel's program walloped Connecticut's champion, who had gone undefeated for eight years.

The checkers player generated much excitement, though it was not perfect and needed help to learn. It was an impressive start. Some years later Patrick Winston, now director of the artificial intelligence laboratory at the Massachusetts Institute of Technology, would work out a program that learned by example, modeled on the way children learn about the world. Show a kid a tree, then 10 trees, and the child will get a feel for "treeness." Winston created Arches, a program to infer "archness" by scrutinizing toy block arches.

More general self-programming systems came during the mid-1970s. Ryszard Michalski at the University of Illinois devised a program that taught itself how to diagnose soybean diseases with 97 percent accuracy. Another impressive self-learning system, Bacon, named after the British philosopher who lived in the late 16th and early 17th centuries, could deduce laws of nature from scientific facts. Given lots of data about the solar system, Bacon independently discovered a law of planetary motion. Using other statements, it figured out a key law of chemical elements.

Some artificial intelligence scientists thumb their noses at Bacon, saying it needs too much help from people to learn. Among those is Douglas Lenat, who as a graduate student at Stanford invented the Automated Mathematician. Loaded with if-then rules and mathematical concepts such as equality, sets and addition, Lenat's creation moved within hours from grade school arithmetic to college mathematics, teaching itself everything. It figured out some 200 theories about numbers, including the idea that some numbers are prime.

Yet Lenat was not satisfied. The Automated Mathematician was locked into the world of mathematics, and he wanted a more general problem solver. In 1976 he launched a more powerful program, Eurisko, the Greek term for "I discover things."

The key to Eurisko was a special pro-

It is said that one computer, translating "the spirit is willing but the flesh is weak" into Russian and back, got "the vodka is good but the meat is rotten."

gramming language that Lenat developed. Called RLL, it allowed a computer to reason more broadly than Lisp, the language the Automated Mathematician used, designed by John McCarthy for strict logical deductions. The new language gave Eurisko the flexibility to deal with all kinds of situations. It could simulate animal evolution, find ways to clean up oil spills, play games, design computer chips — and do many of these tasks as well as humans do.

When Lenat entered Eurisko in a naval war games tournament in 1981, it stole the prize. After another smashing victory in 1982, officials changed the rules, barring it from future competition. Ever intrigued by this sort of thing, the Pentagon found Eurisko fascinating and considered using its strategies to solve real military problems. More recently Lenat has been working on CYC, a common sense-style knowledge base broad enough to read and interpret an entire encyclopedia.

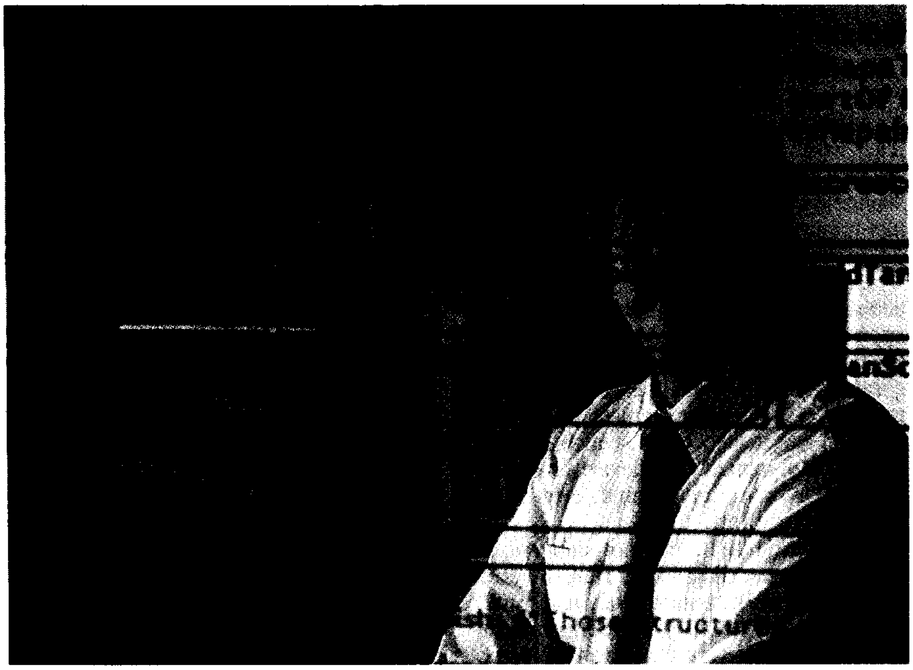
But comprehending natural language — spoken or written, rather than machine code — is a painful process for any computer. Deeply aware of the pitfalls is Roger Schank, head of Yale's artificial intelligence laboratory. Though trained in mathematics and linguistics, his main interest is psychology, which shapes his view of artificial intelligence. Disheartened with logicians' neat approach to language comprehension and often critical of linguistic formality, Schank, a self-described "scruffy," takes a more ragged view. He believes people link words not with meanings, those amorphous abstractions, but with underlying conceptual structures.

For example, from a sentence such as "Bill took Patricia bowling last night," people create little scenarios in their minds to explain the event and then remember the scenario, not the words. Once memorized, the words fall into a coherent image held in the mind. In this scheme, most people, after reading a book or having a conversation, remember its content but rarely its words. They retain the "gist of it." Schank calls these memories "scripts," each script being a little scenario.

In 1974 he and students invented Sam, a story-analyzing program. In one case, a descendant of Sam designed to read news stories, called Cyrus, followed wire service reports during the late 1970s about Secretary of State Cyrus Vance. The program became so adept at Vance trivia that it correctly surmised that Vance's wife had met with the wife of Prime Minister Menachem Begin of Israel at a dinner party, even though that fact was never reported.

Such computer cleverness was exciting. Earlier efforts with natural language had bombed. Even such straightforward tasks as translation bore grotesque fruits. Artificial intelligence folklore has it that one programmer, instructing his computer to translate the English phrase "the spirit is willing but the flesh is weak" into Russian and back again, got a disheartening reply from his blinking terminal: "The vodka is good but the meat is rotten."

The problem of melding syntax and semantics in a computer's so-called mind is still unsolved. Simply parsing sentences, dividing them into grammatical components, just does not work. To understand a sentence, really get its meaning, requires knowledge about the world, about what it is like to be a cognizant human being. Some programs can create persuasive illusions of comprehension, but not the real thing — not yet. One famous program, Eliza, by Joseph Weizenbaum of the Massachusetts Institute of Technology, could mimic the words and disposition of a psychotherapist, talking with a patient:


MCC

Lenat: From problem solvers to a knowledge base that can digest an encyclopedia

Patient: "Men are all alike."

Eliza: "In what way?"

"They're always bugging us about something or other."

"Can you think of a specific example?"

"Well, my boyfriend made me come here."

"Your boyfriend made you come here?"

"He says I'm depressed much of the time."

"I am sorry to hear you are depressed."

Et cetera. But Eliza no more understood the meanings of its statements than anyone else reading this inanity.

Despite efforts by the Pentagon's Defense Advanced Research Projects Agency to develop speech recognition programs — $20 million has gone to research on speech understanding alone — natural language has not yielded well to computers' rigid minds. Hearsay II, the work of Raj Reddy at Carnegie-Mellon University, could converse about scientific documents with a limited vocabulary but not much more. Irus and Proteus, also agency projects, reportedly can talk about certain military situations, with a 4,500-word vocabulary. Yet they are far from perfect.

In a sense, language comprehension programs, plagued by seemingly insurmountable obstacles, have deeply discouraged many in the artificial intelligence community. They argue that, until other areas of cognition are better understood and then modeled, progress in language recognition is doomed to crawl.

About this point, though, there is much disagreement. The optimists, who in each progressive step see an affirmation, argue that these difficulties will succumb to solutions in time.

— *Richard Lipkin*